Penetration Test Report

Linux Multipath TCP

V 1.1
Amsterdam, June 19th, 2024
Public

Document Properties

Client	Linux Multipath TCP
Title	Penetration Test Report
Target	net/mptcp folder of mpTCP Linux kernel development (commit 78d0ce1)
Version	1.1
Pentester	Niek van der Dussen
Authors	Niek van der Dussen, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	June 13th, 2024	Niek van der Dussen	Initial draft
0.2	June 15th, 2024	Marcus Bointon	Review
1.0	June 18th, 2024	Marcus Bointon	1.0
1.1	June 19th, 2024	Niek van der Dussen	Reword future work

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between April 17, 2024 and June 12, 2024 , Radically Open Security B.V. carried out a penetration test for Linux Multipath TCP.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- `net/mptcp` folder of mpTCP Linux kernel development (commit 78d0ce1)

The scoped services are broken down as follows:

- Testing environment setup: 0.5 days
- Code reading: 1.5 days
- Dynamic and static testing: 2 days
- Reporting: 2 days
- **Total effort: 6 days**

1.3 Project objectives

ROS will perform an analysis of the source code of mpTCP with the developers of multipath TCP in order to assess the security of mpTCP in the Linux kernel. To do so, ROS will access the `net/mptcp` folder of mpTCP Linux kernel development (commit `78d0ce1`) and guide the developers of mpTCP in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between April 17, 2024 and June 12, 2024 .

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Low and 3 Unknown-severity issues.

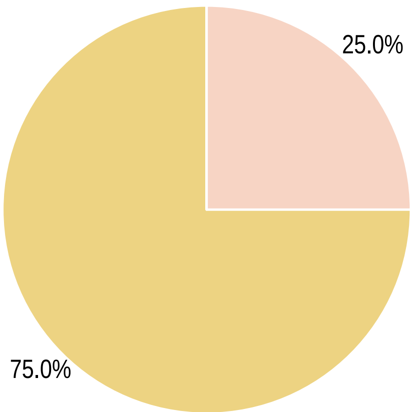
There is much automated testing already in place, but we recommend including static analysis in the pipeline for this project as well. Using a static analyzer (CodeChecker) we were able to find three null pointer dereferences. Unfortunately, we did not have time to confirm whether they were true positives, but it does demonstrate its potential.

If the found issues are true positives and exploitable, an attacker might be able to crash or even exploit the Linux kernel.

1.6 Summary of Findings

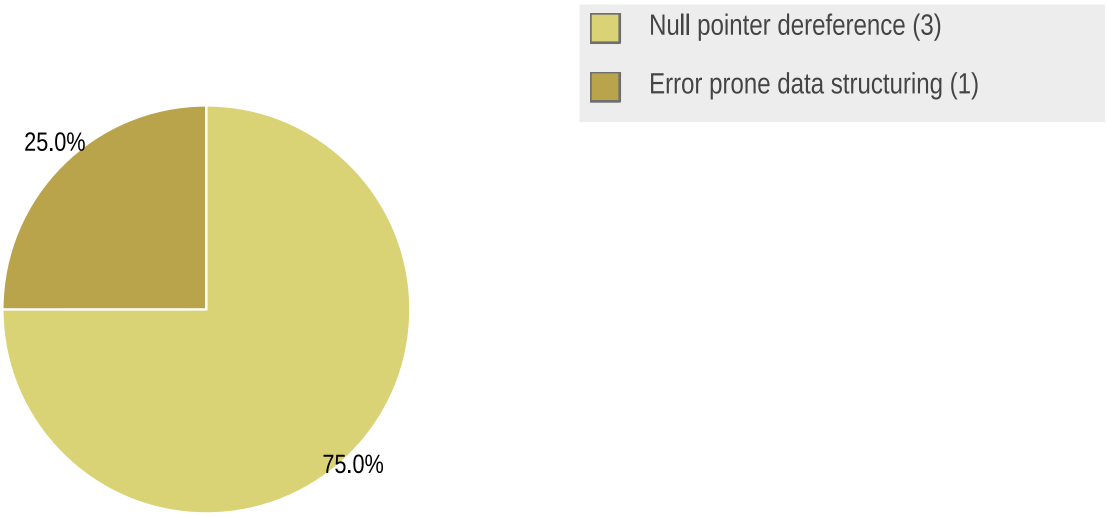
ID	Type	Description	Threat level
CLN-001	Error prone data structuring	Access of data in a pointer is done manually, which seems error-prone. In the case of the nonce, this could lead to using unintended data as the nonce, leading to nonce re-use.	Low
CLN-006	Null pointer dereference	CodeChecker indicates that the pointer ssk in net/mptcp/protocol.c may be dereferenced while being null.	Unknown
CLN-008	Null pointer dereference	CodeChecker indicates that the pointer ssk in net/mptcp/protocol.c may be dereferenced while being null.	Unknown
CLN-009	Null pointer dereference	CodeChecker indicates that the pointer ssk in net/mptcp/protocol.c may be dereferenced while being null.	Unknown

1.6.1 Findings by Threat Level



Low (1)
Unknown (3)

1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-001	Error prone data structuring	Rewriting the code to use more strictly defined forms of accessing parts of a data structure, such as using macros and structs, may introduce new issues. Since the impact of this issue is so low, we do not recommend rewriting it now. However, when the implementation is rewritten anyway, consider ways of accessing data in a structure where the compiler determines the exact address instead of the programmer.
CLN-006	Null pointer dereference	Investigate whether this is a true or false positive. Academic tooling exists for directed fuzzing, using static analysis results such as this issue to guide the fuzzer to confirm this finding as a true positive. However, it might be faster to investigate this finding using the knowledge of the context of mpTCP and following CodeChecker's steps shown in the screenshots in this finding.
CLN-008	Null pointer dereference	See CLN-006
CLN-009	Null pointer dereference	See CLN-006.

2 Methodology

As indicated in the planning, we did this code both by manually inspecting the code and using tools for automation.

The manual inspection was done by reading through the RFC and looking for security properties. Taking these security properties in the design, we looked at how they were implemented. Using this methodology, we found the issue CLN-001 (page 8).

The automated tool we used is called CodeChecker, which is a graphical frontend for static analysis. It has support for several different analyzers, such as cppcheck and LLVM's clang-analyzer. Also, it has converters for outputs of other tools, including "official" kernel development tools such as coccinelle, smatch and sparse. CodeChecker is open source, free to use, and also part of the Visual Studio addon for linux development mentioned in the mptcp docker builder.

Using CodeChecker, we have found the issues CLN-006 (page 9), CLN-008 (page 16), and CLN-009 (page 21).

3 Findings

We have identified the following issues:

3.1 CLN-001 — Hardcoded data structure access

Vulnerability ID: CLN-001

Vulnerability type: Error prone data structuring

Threat level: Low

Description:

Access of data in a pointer is done manually, which seems error-prone. In the case of the nonce, this could lead to using unintended data as the nonce, leading to nonce re-use.

Technical description:

The file `net/mptcp/options.c` has the function

```
static void mptcp_parse_option(const struct sk_buff *skb,
  const unsigned char *ptr, int opsize,
  struct mptcp_options_received *mp_opt)
```

Here, the data at address `ptr` are read as

```
flags = *ptr++;
```

and

```
mp_opt->nonce = get_unaligned_be32(ptr);
ptr += 4;
```

Impact:

This seems error-prone because the data structure must be followed manually, both where the data is written and where it is read. Doing this manually could lead to a security vulnerability in which the nonce could be read as unchanging data, defeating the security purpose of using a nonce. However, such a bug seems unlikely to go unnoticed in practice. Not only must the nonce be read from the incorrect address by the receiver, the transmitter must also have a bug that uses the same incorrect nonce. Without the transmitter having this bug, communication would fail since transmitter and receiver are not using the same nonce.

Recommendation:

Rewriting the code to use more strictly defined forms of accessing parts of a data structure, such as using macros and structs, may introduce new issues. Since the impact of this issue is so low, we do not recommend rewriting it now. However, when the implementation is rewritten anyway, consider ways of accessing data in a structure where the compiler determines the exact address instead of the programmer.

3.2 CLN-006 — Dereference of null pointer protocol.c L1610

Vulnerability ID: CLN-006

Vulnerability type: Null pointer dereference

Threat level: Unknown

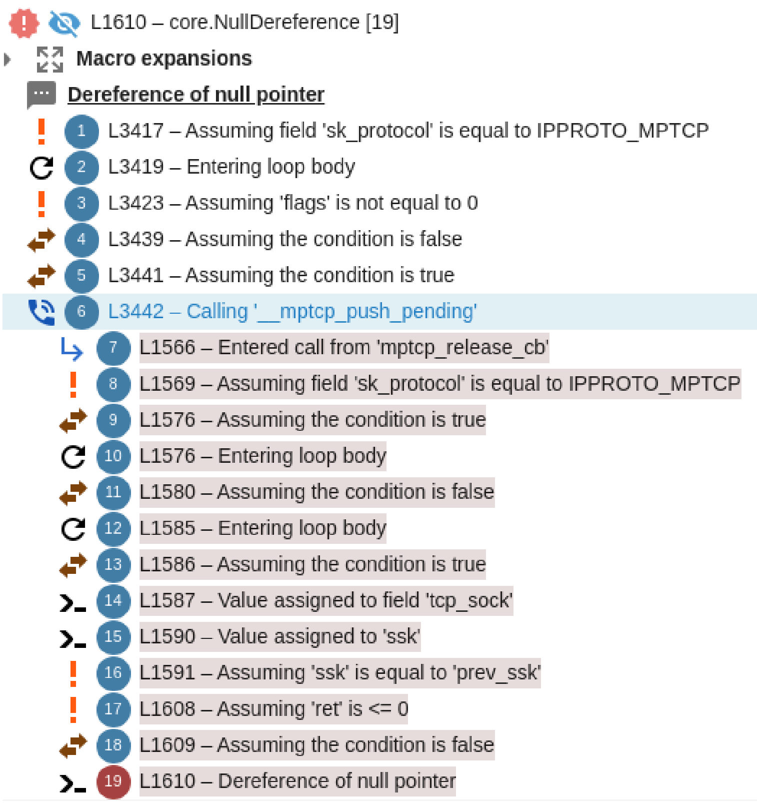
Description:

CodeChecker indicates that the pointer `ssk` in `net/mptcp/protocol.c` may be dereferenced while being null.

Technical description:

According to CodeChecker (in particular, the clang static analyzer `clangsa`) the pointer `ssk` at line 1610 of the file `net/mptcp/protocol.c` can be dereferenced as a null pointer.

The summary of the steps that lead to this error are as follows:



The detailed execution steps are as follows:

(78d0ce14398b088891f3...) | /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```c

/* processes deferred events and flush wmem */
static void mptcp_release_cb(struct sock *sk)
	__must_hold(&sk->sk_lock.slock)
{
	struct mptcp_sock *msk = mptcp_sk(sk);
```

Macro Expansion

```
({typeof (sk )_ptr =(sk );({int __ret_warn_on =!!(_ptr ->sk_protocol
!=IPPROTO_MPTCP );if (__builtin_expect (!!(__ret_warn_on ),0))do
{__auto_type __flags =(1<<0)|(((9)<<8));({asm volatile ("1595"":
nop\n\t"".pushsection .discard.instr_begin\n\t"".long ""1595""b -
.\n\t"".popsection\n\t"::"i"(1595));});do {asm __inline volatile
("1:\t"".byte 0x0f, 0x0b""\n"".pushsection
__bug_table,\"aw\"\n""2:\t"".long ""1b"" - .""\t#
bug_entry::bug_addr\n""\t"".long ""%c0"" - .""\t#
bug_entry::file\n""\t.word %c1""\t# bug_entry::line\n""\t.word
%c2""\t# bug_entry::flags\n""\t.org
2b+%c3\n"".popsection\n""998:\n\t"".pushsection
.discard.reachable\n\t"".long 998b\n\t"".popsection\n\t"::"i"
("net/mptcp/protocol.c"),"i"(3417),"i"(__flags ),"i"(sizeof (struct
bug_entry )));}while (0);({asm volatile ("1596"":
nop\n\t"".pushsection .discard.instr_end\n\t"".long ""1596""b -
.\n\t"".popsection\n\t"::"i"(1596));});}while (0);__builtin_expect
(!!(__ret_warn_on ),0);});_Generic (_ptr ,const typeof (*(_ptr ))*:
((const struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));})),default :((struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));}))));})
```

1 Assuming field 'sk_protocol' is equal to IPPROTO_MPTCP

```c

	for (;;) {
```

2 Entering loop body

```c
		unsigned long flags = (msk->cb_flags & MPTCP_FLAGS_PROCESS_CTX_NEED);
		struct list_head join_list;

		if (!flags)
```

3 Assuming 'flags' is not equal to 0

```c
			break;

		INIT_LIST_HEAD(&join_list);
		list_splice_init(&msk->join_list, &join_list);

		/* the following actions acquire the subflow socket lock
		 *
		 * 1) can't be invoked in atomic scope
```

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
	struct list_head join_list;

	if (!flags)
	  3  Assuming 'flags' is not equal to 0
		break;

	INIT_LIST_HEAD(&join_list);
	list_splice_init(&msk->join_list, &join_list);

	/* the following actions acquire the subflow socket lock
	 *
	 * 1) can't be invoked in atomic scope
	 * 2) must avoid ABBA deadlock with msk socket spinlock: the RX
	 *    datapath acquires the msk socket spinlock while helding
	 *    the subflow socket lock
	 */
	msk->cb_flags &= ~flags;
	spin_unlock_bh(&sk->sk_lock.slock);

	if (flags & BIT(MPTCP_FLUSH_JOIN_LIST))
	  4  Assuming the condition is false
		__mptcp_flush_join_list(sk, &join_list);
	if (flags & BIT(MPTCP_PUSH_PENDING))
	  5  Assuming the condition is true
		__mptcp_push_pending(sk, 0);
	  6  Calling '__mptcp_push_pending'
	if (flags & BIT(MPTCP_RETRANSMIT))
		__mptcp_retrans(sk);
```

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
void __mptcp_push_pending(struct sock *sk, unsigned int flags)
    7 < Entered call from 'mptcp_release_cb' >
{
    struct sock *prev_ssk = NULL, *ssk = NULL;
    struct mptcp_sock *msk = mptcp_sk(sk);
```

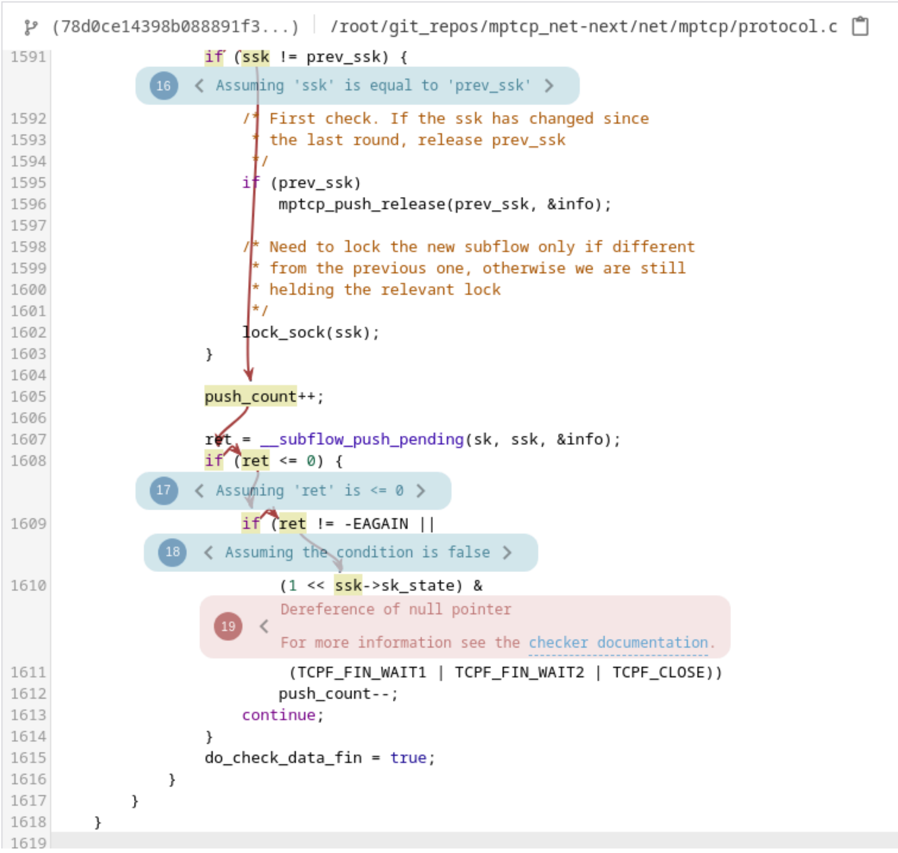
Macro Expansion

```
({typeof (sk )_ptr =(sk );({int __ret_warn_on =!!(_ptr ->sk_protocol
!=IPPROTO_MPTCP );if (__builtin_expect (!!(__ret_warn_on ),0))do
{__auto_type __flags =(1<<0)|(((9)<<8));({asm volatile ("1385"":
nop\n\t"".pushsection .discard.instr_begin\n\t"".long ""1385""b -
.\n\t"".popsection\n\t"::"i"(1385));});do {asm __inline volatile
("1:\t"".byte 0x0f, 0x0b""\n"".pushsection
__bug_table,\"aw\"\n""2:\t"".long ""1b"" - .""\t#
bug_entry::bug_addr\n""\t"".long ""%c0"" - .""\t#
bug_entry::file\n""\t.word %c1""\t# bug_entry::line\n""\t.word
%c2""\t# bug_entry::flags\n""\t.org
2b+%c3\n"".popsection\n""998:\n\t"".pushsection
.discard.reachable\n\t"".long 998b\n\t"".popsection\n\t"::"i"
("net/mptcp/protocol.c"),"i"(1569),"i"(__flags ),"i"(sizeof (struct
bug_entry )));}while (0);({asm volatile ("1386"":
nop\n\t"".pushsection .discard.instr_end\n\t"".long ""1386""b -
.\n\t"".popsection\n\t"::"i"(1386));});}while (0);__builtin_expect
(!!(__ret_warn_on ),0);});_Generic (_ptr ,const typeof (*(_ptr ))*:
((const struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));})),default :((struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));}))));})
```

```
    8 < Assuming field 'sk_protocol' is equal to IPPROTO_MPTCP >
    struct mptcp_sendmsg_info info = {
            .flags = flags,
    };
    bool do_check_data_fin = false;
    int push_count = 1;

    while (mptcp_send_head(sk) && (push_count > 0)) {
        9 < Assuming the condition is true >
        10 < Entering loop body >
        struct mptcp_subflow_context *subflow;
        int ret = 0;

        if (mptcp_sched_get_send(msk))
            11 < Assuming the condition is false >
```

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
	if (mptcp_sched_get_send(msk))
	11 < Assuming the condition is false >
		break;
	push_count = 0;
	mptcp_for_each_subflow(msk, subflow) {
		Macro Expansion
		for (subflow =({void *__mptr =(void *)((&((msk )->conn_list ))->next
		);_Static_assert (__builtin_types_compatible_p (typeof (*((&((msk )-
		>conn_list ))->next )),typeof (((typeof (*subflow )*)0)->node
		))||__builtin_types_compatible_p (typeof (*((&((msk )->conn_list ))-
		>next )),typeof (void )),"pointer type mismatch in container_of()");
		((typeof (*subflow )*)(__mptr -__builtin_offsetof (typeof (*subflow
		),node )));});!list_is_head (&subflow ->node ,(&((msk )->conn_list
		)));subflow =({void *__mptr =(void *)((subflow )->node .next
		);_Static_assert (__builtin_types_compatible_p (typeof (*((subflow )-
		>node .next )),typeof (((typeof (*(subflow ))*)0)->node
		))||__builtin_types_compatible_p (typeof (*((subflow )->node .next
		)),typeof (void )),"pointer type mismatch in container_of()");
		((typeof (*(subflow ))*)(__mptr -__builtin_offsetof (typeof (*
		(subflow )),node )));}))
	12 < Entering loop body >
		if (READ_ONCE(subflow->scheduled)) {
			Macro Expansion
			({do {__attribute__ ((__noreturn__ ))extern void
			__compiletime_assert_1387 (void )__attribute__ ((__error__
			("Unsupported access size for {READ,WRITE}_ONCE().")));if (!((sizeof
			(subflow ->scheduled )==sizeof (char )||sizeof (subflow ->scheduled
			)==sizeof (short )||sizeof (subflow ->scheduled )==sizeof (int
			)||sizeof (subflow ->scheduled )==sizeof (long ))||sizeof (subflow -
			>scheduled )==sizeof (long long )))__compiletime_assert_1387
			();}while (0);(*(const volatile typeof (_Generic ((subflow -
			>scheduled ),char :(char )0,unsigned char :(unsigned char )0,signed
			char :(signed char )0,unsigned short :(unsigned short )0,signed short
			:(signed short )0,unsigned int :(unsigned int )0,signed int :(signed
			int )0,unsigned long :(unsigned long )0,signed long :(signed long
			)0,unsigned long long :(unsigned long long )0,signed long long :
			(signed long long )0,default :(subflow ->scheduled )))*)&(subflow -
			>scheduled ));})
		13 < Assuming the condition is true >
			mptcp_subflow_set_scheduled(subflow, false);
		14 < Value assigned to field 'tcp_sock' >

			prev_ssk = ssk;
			ssk = mptcp_subflow_tcp_sock(subflow);
		15 < Value assigned to 'ssk' >
			if (ssk != prev_ssk) {
			16 < Assuming 'ssk' is equal to 'prev_ssk' >
				/* First check. If the ssk has changed since
				 * the last round, release prev_ssk
				 */
```



Impact:

We didn't investigate whether this is a true positive, which is why we have set the threat level to unknown. If this is a true positive, the null pointer dereference could lead to a crash of the kernel (basics of null pointer dereference here) or even a security vulnerability.

Recommendation:

Investigate whether this is a true or false positive. Academic tooling exists for directed fuzzing, using static analysis results such as this issue to guide the fuzzer to confirm this finding as a true positive. However, it might be faster to investigate this finding using the knowledge of the context of mpTCP and following CodeChecker's steps shown in the screenshots in this finding.

3.3 CLN-008 — Dereference of null pointer protocol.c L2463

Vulnerability ID: CLN-008

Vulnerability type: Null pointer dereference

Threat level: Unknown

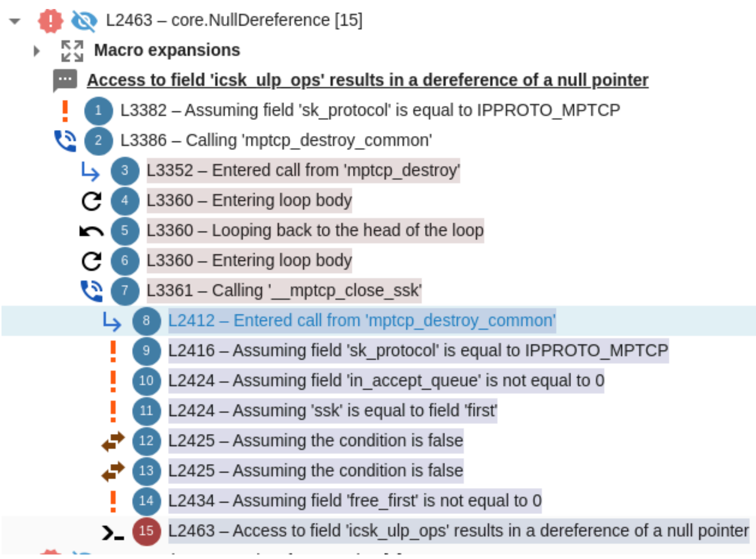
Description:

CodeChecker indicates that the pointer `ssk` in `net/mptcp/protocol.c` may be dereferenced while being null.

Technical description:

According to CodeChecker (in particular, the clang static analyzer `clangsa`) the pointer `ssk` at line 2463 of the file `net/mptcp/protocol.c` can be dereferenced as a null pointer.

The summary of the steps that lead to this error are as follows:



The detailed execution steps are as follows:

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
}

static void mptcp_destroy(struct sock *sk)
{
    struct mptcp_sock *msk = mptcp_sk(sk);
```

Macro Expansion

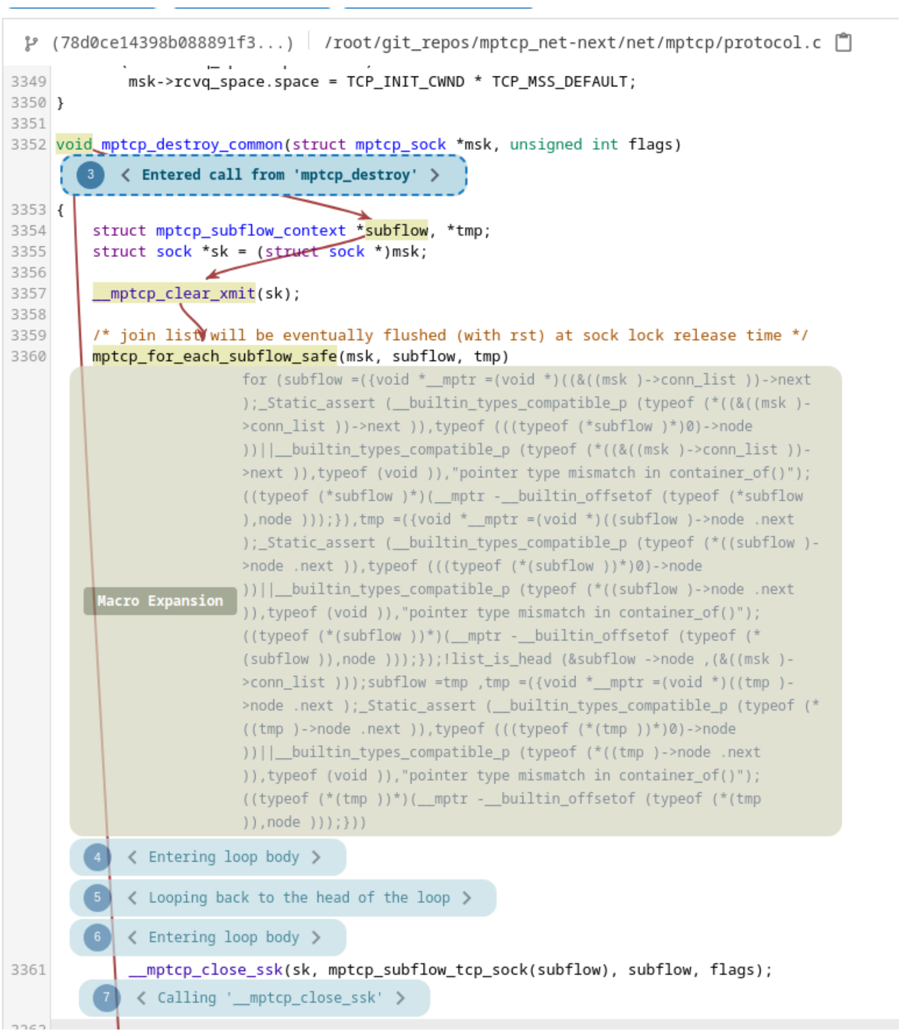
```
({typeof (sk )_ptr =(sk );({int __ret_warn_on =!!(_ptr ->sk_protocol
!=IPPROTO_MPTCP );if (__builtin_expect (!!(__ret_warn_on ),0))do
{__auto_type __flags =(1<<0)|(((9)<<8));({asm volatile ("1589"":
nop\n\t"".pushsection .discard.instr_begin\n\t"".long ""1589""b -
.\n\t"".popsection\n\t"::"i"(1589));});do {asm __inline volatile
("1:\t"".byte 0x0f, 0x0b""\n"".pushsection
__bug_table,\"aw\"\n""2:\t"".long ""1b"" - .""\t#
bug_entry::bug_addr\n""\t"".long ""%c0"" - .""\t#
bug_entry::file\n""\t.word %c1""\t# bug_entry::line\n""\t.word
%c2""\t# bug_entry::flags\n""\t.org
2b+%c3\n"".popsection\n""998:\n\t"".pushsection
.discard.reachable\n\t"".long 998b\n\t"".popsection\n\t"::"i"
("net/mptcp/protocol.c"),"i"(3382),"i"(__flags ),"i"(sizeof (struct
bug_entry )));}while (0);({asm volatile ("1590"":
nop\n\t"".pushsection .discard.instr_end\n\t"".long ""1590""b -
.\n\t"".popsection\n\t"::"i"(1590));});}while (0);__builtin_expect
(!!(__ret_warn_on ),0);});_Generic (_ptr ,const typeof (*(_ptr ))*:
((const struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));})),default :((struct mptcp_sock *)({void *__mptr =(void *)(_ptr
);_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
)),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
)),"pointer type mismatch in container_of()");((struct mptcp_sock *)
(__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
)));})));})
```

1 Assuming field 'sk_protocol' is equal to IPPROTO_MPTCP

```

    /* allow the following to close even the initial subflow */
    msk->free_first = 1;
    mptcp_destroy_common(msk, 0);
```

2 Calling 'mptcp_destroy_common'

```
    sk_sockets_allocated_dec(sk);
}
```



(78d0ce14398b088891f3...)
/root/git_repos/mptcp_net-next/net/mptcp/protocol.c
msk->rcvq_space.space = TCP_INIT_CWND * TCP_MSS_DEFAULT;
}
void mptcp_destroy_common(struct mptcp_sock *msk, unsigned int flags)
3
Entered call from 'mptcp_destroy'
{
struct mptcp_subflow_context *subflow, *tmp;
struct sock *sk = (struct sock *)msk;
__mptcp_clear_xmit(sk);
/* join list will be eventually flushed (with rst) at sock lock release time */
mptcp_for_each_subflow_safe(msk, subflow, tmp)
Macro Expansion
for (subflow =({void *__mptr =(void *)((&((msk)->conn_list))->next);_Static_assert (__builtin_types_compatible_p (typeof (*((&((msk)->conn_list))->next)),typeof (((typeof (*subflow)*)0)->node))||__builtin_types_compatible_p (typeof (*((&((msk)->conn_list))->next)),typeof (void)),"pointer type mismatch in container_of()");((typeof (*subflow)*)(__mptr -__builtin_offsetof (typeof (*subflow),node)));}),tmp =({void *__mptr =(void *)((subflow)->node .next);_Static_assert (__builtin_types_compatible_p (typeof (*((subflow)->node .next)),typeof (((typeof (*(subflow))*)0)->node))||__builtin_types_compatible_p (typeof (*((subflow)->node .next)),typeof (void)),"pointer type mismatch in container_of()");((typeof (*(subflow))*)(__mptr -__builtin_offsetof (typeof (*(subflow)),node)));});!list_is_head (&subflow ->node ,(&((msk)->conn_list)));subflow =tmp ,tmp =({void *__mptr =(void *)((tmp)->node .next);_Static_assert (__builtin_types_compatible_p (typeof (*((tmp)->node .next)),typeof (((typeof (*(tmp))*)0)->node))||__builtin_types_compatible_p (typeof (*((tmp)->node .next)),typeof (void)),"pointer type mismatch in container_of()");((typeof (*(tmp))*)(__mptr -__builtin_offsetof (typeof (*(tmp)),node)));}))
4
Entering loop body
5
Looping back to the head of the loop
6
Entering loop body
__mptcp_close_ssk(sk, mptcp_subflow_tcp_sock(subflow), subflow, flags);
7
Calling '__mptcp_close_ssk'

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
 */
static void __mptcp_close_ssk(struct sock *sk, struct sock *ssk,
    8  < Entered call from 'mptcp_destroy_common' >
                              struct mptcp_subflow_context *subflow,
                              unsigned int flags)
{
    struct mptcp_sock *msk = mptcp_sk(sk);
        Macro Expansion
        ({typeof (sk )_ptr =(sk );({int __ret_warn_on =!!(_ptr ->sk_protocol
        !=IPPROTO_MPTCP );if (__builtin_expect (!!(__ret_warn_on ),0))do
        {__auto_type __flags =(1<<0)|(((9)<<8));({asm volatile ("1456"":
        nop\n\t"".pushsection .discard.instr_begin\n\t"".long ""1456""b -
        .\n\t"".popsection\n\t"::"i"(1456));});do {asm __inline volatile
        ("1:\t"".byte 0x0f, 0x0b""\n"".pushsection
        __bug_table,\"aw\"\n""2:\t"".long ""1b"" - .""\t#
        bug_entry::bug_addr\n""\t"".long ""%c0"" - .""\t#
        bug_entry::file\n""\t.word %c1""\t# bug_entry::line\n""\t.word
        %c2""\t# bug_entry::flags\n""\t.org
        2b+%c3\n"".popsection\n""998:\n\t"".pushsection
        .discard.reachable\n\t"".long 998b\n\t"".popsection\n\t"::"i"
        ("net/mptcp/protocol.c"),"i"(2416),"i"(__flags ),"i"(sizeof (struct
        bug_entry )));}while (0);({asm volatile ("1457"":
        nop\n\t"".pushsection .discard.instr_end\n\t"".long ""1457""b -
        .\n\t"".popsection\n\t"::"i"(1457));});}while (0);__builtin_expect
        (!!(__ret_warn_on ),0);});_Generic (_ptr ,const typeof (*(_ptr ))*:
        ((const struct mptcp_sock *)({void *__mptr =(void *)(_ptr
        );_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
        )),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
        ))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
        )),"pointer type mismatch in container_of()");((struct mptcp_sock *)
        (__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
        )));})),default :((struct mptcp_sock *)({void *__mptr =(void *)(_ptr
        );_Static_assert (__builtin_types_compatible_p (typeof (*(_ptr
        )),typeof (((struct mptcp_sock *)0)->sk .icsk_inet .sk
        ))||__builtin_types_compatible_p (typeof (*(_ptr )),typeof (void
        )),"pointer type mismatch in container_of()");((struct mptcp_sock *)
        (__mptr -__builtin_offsetof (struct mptcp_sock ,sk .icsk_inet .sk
        )));})));})
        9  < Assuming field 'sk_protocol' is equal to IPPROTO_MPTCP >
    bool dispose_it, need_push = false;

    /* If the first subflow moved to a close state before accept, e.g. due
     * to an incoming reset or listener shutdown, the subflow socket is
     * already deleted by inet_child_forget() and the mptcp socket can't
     * survive too.
     */
    if (msk->in_accept_queue && msk->first == ssk &&
        10  < Assuming field 'in_accept_queue' is not equal to 0 >
        11  < Assuming 'ssk' is equal to field 'first' >
        (sock_flag(sk, SOCK_DEAD) || sock_flag(ssk, SOCK_DEAD))) {
        12  < Assuming the condition is false >
        13  < Assuming the condition is false >
```

(78d0ce14398b088891f3...) /root/git_repos/mptcp_net-next/net/mptcp/protocol.c

```
        /* ensure later check in mptcp_worker() will dispose the msk */
        sock_set_flag(sk, SOCK_DEAD);
        mptcp_set_close_tout(sk, tcp_jiffies32 - (mptcp_close_timeout(sk) + 1));
        lock_sock_nested(ssk, SINGLE_DEPTH_NESTING);
        mptcp_subflow_drop_ctx(ssk);
        goto out_release;
    }

    dispose_it = msk->free_first || ssk != msk->first;
                 14  < Assuming field 'free_first' is not equal to 0 >
    if (dispose_it)
        list_del(&subflow->node);

    lock_sock_nested(ssk, SINGLE_DEPTH_NESTING);

    if ((flags & MPTCP_CF_FASTCLOSE) && !__mptcp_check_fallback(msk)) {
        /* be sure to force the tcp_close path
         * to generate the egress reset
         */
        ssk->sk_lingertime = 0;
        sock_set_flag(ssk, SOCK_LINGER);
        subflow->send_fastclose = 1;
    }

    need_push = (flags & MPTCP_CF_PUSH) && __mptcp_retransmit_pending_data(sk);
    if (!dispose_it) {
        __mptcp_subflow_disconnect(ssk, subflow, flags);
        release_sock(ssk);

        goto out;
    }

    subflow->disposable = 1;

    /* if ssk hit tcp_done(), tcp_cleanup_ulp() cleared the related ops
     * the ssk has been already destroyed, we just need to release the
     * reference owned by msk;
     */
    if (!inet_csk(ssk)->icsk_ulp_ops) {
      Macro Expansion  _Generic (ssk ,const typeof (*(ssk ))*:((const struct
                       inet_connection_sock *)({void *__mptr =(void *)(ssk );_Static_assert
                       (__builtin_types_compatible_p (typeof (*(ssk )),typeof (((struct
                       inet_connection_sock *)0)->icsk_inet .sk
                       ))||__builtin_types_compatible_p (typeof (*(ssk )),typeof (void
                       )),"pointer type mismatch in container_of()");((struct
                       inet_connection_sock *)(__mptr -__builtin_offsetof (struct
                       inet_connection_sock ,icsk_inet .sk )));})),default :((struct
                       inet_connection_sock *)({void *__mptr =(void *)(ssk );_Static_assert
                       (__builtin_types_compatible_p (typeof (*(ssk )),typeof (((struct
                       inet_connection_sock *)0)->icsk_inet .sk
                       ))||__builtin_types_compatible_p (typeof (*(ssk )),typeof (void
                       )),"pointer type mismatch in container_of()");((struct
                       inet_connection_sock *)(__mptr -__builtin_offsetof (struct
                       inet_connection_sock ,icsk_inet .sk )));})))
      15  <  Access to field 'icsk_ulp_ops' results in a dereference of a null pointer
             For more information see the checker documentation.
```

Impact:

See CLN-006 (page 9)

Recommendation:

See CLN-006 (page 9)

3.4 CLN-009 — Dereference of null pointer protocol.c L2392

Vulnerability ID: CLN-009

Vulnerability type: Null pointer dereference

Threat level: Unknown

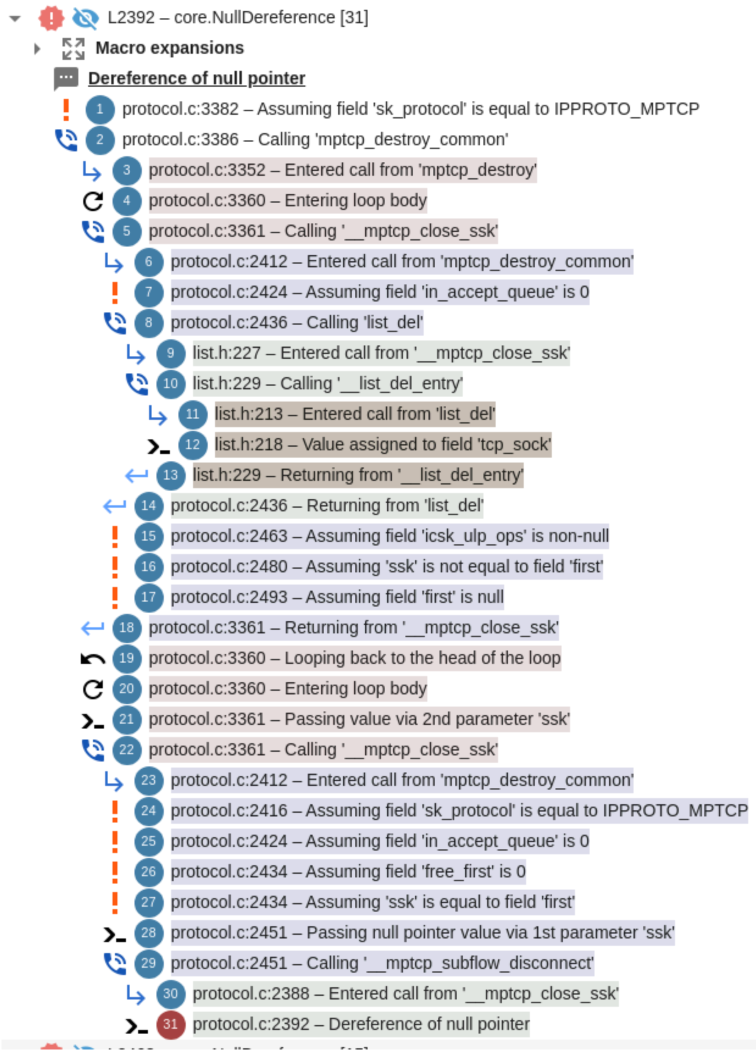
Description:

CodeChecker indicates that the pointer `ssk` in `net/mptcp/protocol.c` may be dereferenced while being null.

Technical description:

According to CodeChecker (in particular, the clang static analyzer `clangsa`) the pointer `ssk` at line L2392 of the file `net/mptcp/protocol.c` can be dereferenced as a null pointer.

The summary of the steps that lead to this error are as follows:



Unlike CLN-006 (page 9) and CLN-008 (page 16), we won't show the step by step here because many of the 31 steps are function calls. For a detailed view, see the HTML output file of CodeChecker attached to this report.

Impact:

See CLN-006 (page 9).

Recommendation:

See CLN-006 (page 9).

4 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.1 NF-003 — Build for multiple architectures

We built the mpTCP development version of the Linux kernel using the project's build instructions. This worked for x86 architectures, but not for aarch64. That issue was fixed during this audit. We intended to build the kernel on less-common, and therefore potentially less-tested architectures, to try to find bugs in the automated tests for mpTCP. However, we learned from one of the mpTCP developers that later in the upstreaming process, tests of exactly this nature are done automatically. In particular, a recent Intel test run ran on 24 architectures with 139 different configurations.

Given how much testing of this kind is already in place, we decided our time would be better spent on other topics.

4.2 NF-007 — Z3 for CodeChecker

Static analyzers can give many false positives, which is most likely the case here as well. Taking only the static analysis results from mpTCP (see Methodology TODO link to section), CodeChecker still gives us 3885 results.

The Z3 Theorem Prover is an experimental feature of CodeChecker to reduce the number of false positives. Using it however, is not straightforward. Most importantly, you need to build Clang yourself with Z3 enabled. We first attempted this on Debian 12, but this was not feasible because it is shipped with GCC 12, which is too low for CodeChecker if you want to use the GCC Static Analyzer backend. Using Fedora 40 we got the Z3 functionality to work. The hope was that Z3 could be leveraged to reduce the number of false positives, but CodeChecker still got 3885 results.

There is also an option to use Z3 as the only backend for CodeChecker as described here, but we could not get this to work. Also, note that is slower and may even hang if no timeout is set, so this is not recommended for automated purposes.

For reproducibility of analyzing in CodeChecker with Z3 in Fedora 40, we will share our steps here.

```
dnf install -y dnf-plugins-core git cmake gcc gcc-c++ autoconf automake unzip python3 python3-devel
 cppcheck clang-tools-extra
mkdir ~/git_repos
cd ~/git_repos
git clone https://github.com/llvm/llvm-project.git
cd llvm-project
git checkout llvmorg-18.1.7
mkdir build
cd build
# Download the library directly, using `dnf install` results in the library not being found by cmake
wget  https://github.com/Z3Prover/z3/releases/download/z3-4.13.0/z3-4.13.0-x64-glibc-2.35.zip
unzip z3-4.13.0-x64-glibc-2.35
cmake -DLLVM_Z3_INSTALL_DIR=.  -DLLVM_ENABLE_Z3_SOLVER=1 -DLLVM_ENABLE_PROJECTS=clang -
DCMAKE_BUILD_TYPE=Release -G "Unix Makefiles" ../llvm
# build Clang with Z3 enabled
```

```
make
# install our newly build Clang with z3 enabled
cp build/bin* /usr/local/bin/

# clone the mptcp repo
cd ~/git_repos
git clone https://github.com/multipath-tcp/mptcp_net-next.git
cd mptcp_net-next
# optional: checkout the version that we used throughout this audit
git checkout 78d0ce14398b088891f34b2c83c2e4b650f334fc

#verify that we are using our clang version, the following output should be our install location `/
usr/local/bin`
which clang

# build the linux kernel using our Clang with Z3 enabled
docker run -e INPUT_CLANG=1 -v "${PWD}:${PWD}:rw" -w "${PWD}" -v "${PWD}/.home:/root:rw" --rm   -it
 --privileged --pull always mptcp/mptcp-upstream-virtme-docker:latest manual
# leave container, ctrl+d

# we now have a file with all build commands used by clang while building this repo. We're going to
 trim it down to focus only on mptcp
docker run -e INPUT_CLANG=1 -v "${PWD}:${PWD}:rw" -w "${PWD}" -v "${PWD}/.home:/root:rw" --rm   -it
 --privileged --pull always mptcp/mptcp-upstream-virtme-docker:latest cmd bash
# Run the following in the docker itself
jq 'map(select(.file | contains ("/mptcp/")))' .virtme/build-clang/compile_commands.json >
 compile_commands-mptcp.json
# leave the docker
exit

# still in the mptcp repo
# Install CodeChecker
python3 -m venv .venv
source .venv/bin/activate
pip install codechecker setuptools

# run CodeChecker
CodeChecker analyze compile_commands-mptcp.json --z3-refutation on --enable sensitive --enable
 portability --output .codechecker/reports
CodeChecker server &
# push them to our local server
CodeChecker store .codechecker/reports/ -n mptcp
# in your browser, go to localhost:8081 to use the CodeChecker Web UI
# or export as HTML. Less user friendly, much easier to share
CodeChecker store .codechecker/reports/ -n mptcp
```

5 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.
- **Verifying that implementation follows design**
  We recommend going through the protocol design and identifying security properties of the protocol. Next, check whether these security properties are correctly implemented.
  As part of this audit, we checked the implementation of one security principle in particular, namely whether each nonce is randomly generated. The fact that this was quite difficult to verify, lead to us reporting this difficulty as a point of attention (see CLN-001 (page 8)). This demonstrates that the exercise of verifying that the security properties of the implementation follow the design, has the potential to uncover other implementation errors.
- **Kernel fuzzing**
  There is already fuzzing in place using syzkaller, run by the bot called syzbot. Its results for the mpTCP part can be seen here, by clicking through `net\mptcp`. Inspecting coverage could be useful for finding functions that aren't covered, and making sure they are tested (either manually or with manual effort and other fuzzers such as AFL++).

6 Conclusion

We discovered 1 Low and 3 Unknown-severity issues during this penetration test.

Most of the "low-hanging fruit" for security and stability has already been dealt with for this project. The developers are experienced and knowledgeable, and security has been taken seriously from the outset. This is demonstrated by the fact that threat modelling and security considerations are part of the mpTCP RFC. Additionally, this project is functionally tested and fuzzed by other members of the Linux ecosystem, strengthening our faith in the project's security. The team is already aware that there may be insights left to gain from static analysis, even though there are a very large number of probable false positives. Given that there may still be true positives hiding in this haystack, we recommend looking into tools to better visualize the findings and tracking marked false positives. During this audit, it seemed that CodeChecker could be a tool to achieve this goal.

The protocol design has already been audited in the past, but we nevertheless recommend (manually) verifying that the implementation actually follows the protocol design. For example, a nonce is assumed in the protocol design to be truly used only once. However, it is not trivial to spot a nonce being reused in the implementation. Looking for more security properties (such as the use of nonces, or whether they are generated randomly) in the protocol design and verifying their correct implementation could prove advantageous.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Niek van der Dussen	Niek is a pentester with several years of experience in embedded system development, a bachelor's degree in electrical engineering and a master's degree in computer science. He has always had a special interest in security, and practical security experience as a developer. Niek is currently expanding his skills as an all-round security specialist by doing the PEN-200 OSCP course.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.